10627325 8-26-03

The Patent Office

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed K Behen

Dated 24 July 2003

atents Form 1/77

Patents Act 1977
(Rule 16)

THE PATENT OFFICE
K
24 JUL 2002
LONDON
The Patent Office

25JUL02 E735800-2 002629
P01/7700 0.00-0217201.3

1/77

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP10 8QQ

1. Your reference — J00044620GB

2. Patent application number
(The Patent Office will fill in this part)
0217201.3
24 JUL 2002

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

or of

Beach Solutions Limited
Suite 21, Atlas House
West Devon Business Park
Tavistock, Devon
PL19 9DP
United Kingdom

Patents ADP number *(if you know it)*

8223447,8001

If the applicant is a corporate body, give the country/state of its incorporation — United Kingdom

4. Title of the invention — XML Database Differencing Engine

5. Name of your agent *(if you have one)* — RGC Jenkins & Co.

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

26 Caxton Street
London SW1H ORJ
United Kingdom

Patents ADP number *(if you know it)* — 03966736001

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications *(and if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month/ year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month/ year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request *(Answer 'Yes' if:*

a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

Yes

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	None
Description	Nineteen (19)
Claim(s)	None
Abstract	None
Drawing(s)	None

10. If you are also filing any of the following, state how many against each item.

Priority documents	None
Translation of priority documents	None
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	None
Request for preliminary examination and search *(Patents Form 9/77)*	None
Request for substantive examination *(Patents Form 10/77)*	None
Any other documents *(please specify)*	None

11. I/We request the grant of a patent on the basis of this application.

Signature R.G.C. Jenkins & Co Date 24 July, 2002

R.G.C. JENKINS & CO

12. Name and daytime telephone number of person to contact in the United Kingdom

Mark Milhench 020-7931-7141

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay, please contact the Patent Office.*

Beach Solutions

PATENT INFORMATION

XML Database Differencing Engine

Document: BSPTAP.209.10.1.D

Contents

1 Terms, Definitions and Abbreviations

[Those in italics will probably not be needed.]

ADO: *ActiveX Data Object(s)*

Class: In object-oriented programming a definition of a class or type of object describing the attributes of objects represented by that class or type and the operations that these objects can perform or have performed upon them

COM: *Component Object Model*

CSV: Comma Separated Value: A file format in which values on a line are separated by comma characters; importable directly into the cell rows of most spreadsheet applications, eg. Microsoft Excel, Lotus 123, Borland Quattro, SuperCalc, etc

DAO: *Data Access Object(s)*

Database: A document, file or collection of files storing a set of data in a structured way in conformance with a defined plan known as a schema

DB: Database

DBMS: Database Management System

GUI: Graphical User Interface

HTML: HyperText Markup Language

ID: Identity (or, in some contexts, identify or identification)

Interface: In object-oriented programming the set of methods defined for a particular object class

Method: In object-oriented programming an operation that an object can perform or have performed upon it

Normalisation: An operation, carried out during the analysis of data to be represented in a database and hence in the development and definition of a schema, to avoid unnecesary duplication of data ('one fact in one place') and promote rapid access for storage and retrieval of data by providing efficient search paths. See note below on Normalisation.

Object: In object-oriented programming an instance of a class in which the attributes have particular values for that instance (and hence differentiate it from other objects)

Parser: A tool which automatically extracts data from text files

Path: An ordered, directed, chain of characteristics (eg. names or IDs), locations, links or relationships, etc

Platform: A host computer together with its operating system

RDB: A relational database – a database holding not only the data but the relationships between them

RDBMS: Relational DBMS

RTF: Rich Text Format: A textual file format where body text is interspersed with formatting codes; originating from Microsoft and compatible with Microsoft Word and Adobe Framemaker

Schema: A plan for a database describing the data it may contain, their types, formats, representation and relationships

Database Schema: The document set consisting of the entity model (usually an entity-relationship diagram) and the lists of entity attributes

SQL: Structured Query Language: A general purpose non-proprietary language for retrieving data from and updating RDBs in a structured way using compound expressions and complex operations

TSV: Tab Separated Value: A file format in which values on a line are separated by tab characters; useful for plain text editors

Unix: One of a 'family' of similar operating systems based on the original Unix (eg. SunOS & Solaris from Sun Microsystems, HP Unix from Hewlett-Packard, various offerings of Linux, etc)

Windows: One of the family of personal computer operating systems provided by The Microsoft Corporation (eg. Windows 3.1/95/NT/98/2000/ME/XP)

XML: Extensible Markup Language – A subset of SGML (Standard Generalised Markup Language) used for representing the structure of and the relationships between elements of data in a textually formatted document

2 Introduction: Quantification of Change

Databases are used in many organisations to store data in an ordered and structured manner. Although many manual database systems exist the trend is towards the use of electronically stored and manipulated databases.

In real-world situations users regularly need to know precisely the detailed differences between the content of two large and complex databases, most often two different versions of the same database, in terms of the data values they contain.

Generally, the data are stored in a proprietary binary format underlying a database management tool normally referred to as a database management system (DBMS), examples of which include Microsoft Access & SQL Server, Informix, Oracle, Ingress, DBase, and Borland Paradox. These examples also happen to be a particular form of DBMS known as a relational DBMS (RDBMS) because there are distinct and significant advantages in using this type of system. This document is concerned only with databases of the type managed by RDBMSs: relational databases (RDBs).

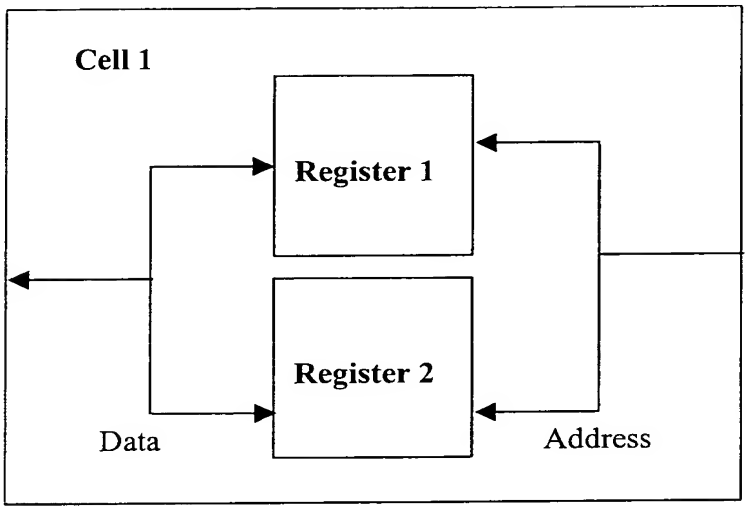
Access to the data in an RDB normally requires use of either the RDBMS itself or an application ('tool') using the RDBMS' application program interface (API) [possibly using a structured query language (SQL)] where such an API is provided. In the first case a specific tool needs to be provided for each proprietary database format. In both cases, a detailed knowledge of the database schema and a tool tailored to suit it are required.

For a generic approach, a generic language or format to describe, quantify and handle the data and a generic means of dealing with the schema-dependent relationships are needed. This patent application addresses such an approach.

..... ?????

2.1 Terminology

A short note on terminology may be helpful in understanding what follows. Examples will be used which rest upon the idea that most designs are hierarchical and use a model often found in electronic engineering that consists of a system containing one or more sub-systems recursively. Our example is a simple one consisting of a major block (which could be a re-usable block of Intellectual Property that would be inserted as a sub-system into a system) and its component parts (which are further sub-systems of that block). The particular example below is a block of structured information called a cell object which contains a number of register objects (locations) which happen severally to contain numerically codified data. Each of these object types has various characteristics, such as the numbers of data wires and address wires (what are called data and address widths) and whether or not it is possible to write numbers to or read them from the registers.



2.1.1 Databases

A database table represents an entity type and lists all entities of that type as records. An entity essentially corresponds to an object in the real world, an entity type thus corresponding to an object type. Records contain fields and fields correspond to attributes or characteristics of an object. (The term characteristic will be used here for an object's attribute to distinguish it from the special meaning of attribute in XML as described later.) Any particular object will have its own characteristic values which distinguish it from other objects of the same type (or may not, as the case may be). In order reliably to distinguish between objects having identical sets of characteristic values, an additional field known as a key field is reserved for a value known as a key value that is unique to that object within that table.

An example is given below of a table for objects of type **Cell** having six characteristics. Three instances are given each having a different set of values for the characteristic fields in its record. The field named **id** is the key field in this case.

Table Example:

Cell

id	name	version	dataWidth	addressWidth	d scription
1	adder	1	8	32	adds two 8-bit values
2	multiplier	0	16	32	multiplies two 8-bit values
3	accumulator	2	16	32	accumulates results

2.1.2 XML

Essentially, XML files represent databases as collections of objects known as 'elements', using tags to name and identify them. Elements can have 'child' elements and the ensuing hierarchy can be represented either explicitly, where parental elements contain (or 'embody') their children, or implicitly, where children are described separately and identify their parents by 'referential attributes'. All elements may have 'attributes' representing the characteristics of their associated objects.

A typical XML file might represent the first instance of a cell from the above table as follows. It also contains an instance of a register (which would appear in a **Register** table). Here the register is a component or child element of the cell with **id**=1 and identifies its parent by the reference cellId="cell_1"; ie. referentially (implicitly) and not by containment (explicitly).

Note that the <xxx yy="z"> and </xxx> are XML tags known as the start and end tags and form an XML element which represents a database object instance. Where a tagged parent has no children and only XML attributes, a combined (shorthand) tag format is used, eg. <name value="adder"/>.

Note also, in this example the **id** object characteristic is represented by an XML attribute 'id' within the Cell tag but that the other object characteristics are represented by child elements, each holding the value of the object characteristic in an XML attribute with the name 'value'.

1st XML Example:

```
<Cell id="cell_1">
        <name value="adder"/>
        <version value="1"/>
        <dataWidth value="8"/>
        <addressWidth value="32"/>
        <description value="adds two 8-bit values"/>
</Cell>

<Register id="register_1" cellId="cell_1">
        <name value="Addend"/>
        <accessType value="readWrite"/>
        <dataWidth value="8"/>
        <description value="holds first item to be added"/>
</Register>

<Register id="register_2" cellId="cell_1">
        <name value="Augend"/>
        <accessType value="readWrite"/>
        <dataWidth value="8"/>
        <description value="holds second item to be added"/>
</Register>
```

This is only one way of representing the information in XML. The following example achieves the same thing but by using containment of the child object by the parent object and representing the object characteristics directly by XML attributes. Note that no referential attribute is needed in the register tag to identify its parent cell.

2nd XML Example:

```
<Cell
        id="cell_1"
        name value="adder"
        version value="1"
        dataWidth value="8"
        addressWidth value="32"
        description value="adds two 8-bit values"
        />
        <Register
                id="register_1"
                name value="Added"
                accessType value="readWrite"
                dataWidth value="8"
                description value="holds first item to be added"
        />
        <Register
                id="register_2"
                name value="Augend"
                accessType value="readWrite"
                dataWidth value="8"
                description value="holds second item to be added"
        />
</Cell>
```

The first form is much easier to generate automatically from a conventional relational database and to parse for input into an application. This form has been adopted for the particular implementation described later.

Within this implementation, object characteristics are actually represented by child elements each of whose attribute 'value' contains the particular value of the characteristic. In summary, the correspondence between objects and their characteristics on the one hand and XML elements and their attributes on the other is shown in the following table.

Table of Object-Element Relationships in this Implementation:

Object Domain	XML Domain
Object type (table name)	Parent element tag name
Object (table record)	Parent element
Identifying & referential characteristics	Parent element attributes
Object characteristic	Child element
Object characteristic name	Child element tag name
Object characteristic value	Child "value"-attribute value

2.2 *A Priori* Assumptions

In an RDB each and every item of data (each and every datum) is typically held in a field within a record (where a record is held within a table) and can be considered in isolation. The content of the database can thus be viewed as an hierarchy of data collections where the hierarchical information is represented by the database schema. Other information such as the characteristics/attributes/types/formats of each datum are also represented by the schema.

In differencing two databases it is taken to be reasonable to assume that they conform with the same schema, otherwise one is not comparing like with like. Also, where the schema represents information about a real-world physical object it is arguably reasonable to assume that the relational hierarchy can be represented by a tree or collection of trees.

These assumptions are made in the approach taken.

2.3 Background: The Need to Quantify Change

2.3.1 Data Changes *per se*

During the life-cycle of a design, changes are usually made to the stored data values. This may be, for instance, as the design evolves, corrections are made, or variants are required.

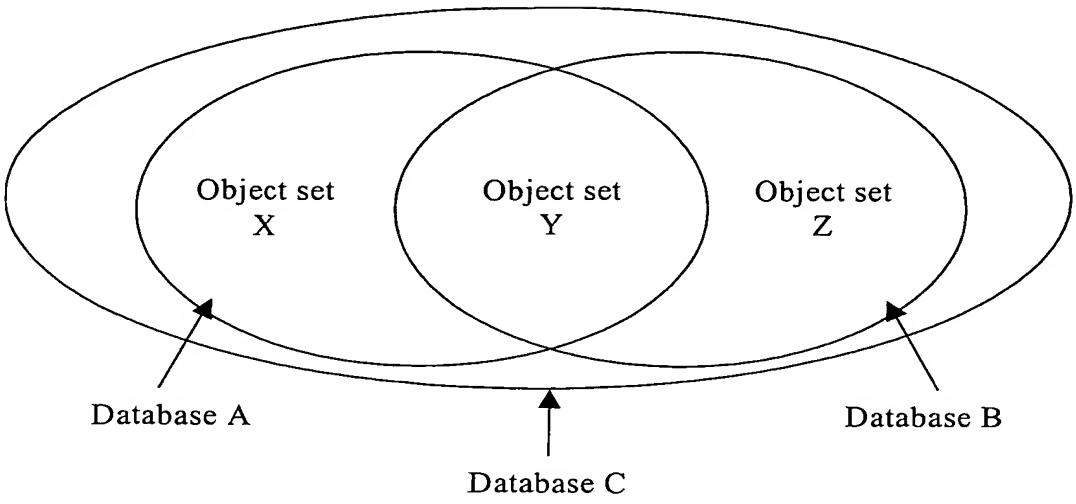
2.3.2 Schema Changes

Entity types, data types and relationships are often added as the design schema grows.

These schema changes are only acceptable provided they are additive and the application is compatible with the super-set. This needs a little explanation.

Referring to the diagram below, suppose that initially the database schema is capable of representing a database A containing sets X and Y of object types and that, later, representation of a set Z of object types is added to the schema. This is fine as long as addition of the set Z of objects types doesn't invalidate the integrity and normalisation of the previous version of the schema and that the whole (new version of the) schema remains integral and fully-normalised. By integral we mean that any set of objects within the database in question is a valid set, all children having parents and there being no missing or broken referential links.

If this is the case, it is possible to compare (and hence differentiate) any two databases conforming to the whole schema, even though one or both of the databases may also conform to the previous schema version. In other words, it is possible to compare databases which individually contain objects from any one or more of the sets X, Y, and Z. For example, one can validly compare any two of databases A, B, and C. Further, neither or both of a database pair may contain set Y. This can be summed up by saying that the two databases of the pair may consist of sets that intersect to any degree, completely, or not at all, and still be validly compared.



2.3.3 Management of Change

Management of change requires that from time to time versions of the database are saved or archived (along with schematic information), eg. for project control, record, or other purposes. As part of the change management process, data changes need to be verified and accounted. This implies a requirement for a means of determining absolutely and accurately the differences between (any two) versions of a database, and for recording when and under what circumstances the comparison was performed.

[The following issue is essentially what should be addressed in a 'Cross-View' patent and not included here.

> *"Additionally, for groups of designers handling different aspects of the design, eg. software, hardware, verification, synthesis, etc, different 'views' or abstractions of the data are normally generated to serve their particular purposes. Some mechanism is required for comparison of these views or abstractions to ensure that no inconsistencies have arisen either in their generation or their source data. It may well be that they need to be compared one with another or with the source database itself."*
>
> *"Provided that a view can be parsed and the data entered into the appropriate part(s) of the database then that view may be compared with another view or database."*

]

[Other patents should be raised for certain associated Beach Solutions proprietary items, eg:

Easi-Studio
Schema
Schema Object
Script Services
Generator Core
Generation Process
Generator Kit

]

3 The Problem: Issues to be Addressed

There are several issues that need to be addressed when considering the problems surrounding the comparison of databases with a view to determining the detailed differences between them. These issues boil down to a set of implied requirements. The issues are as follows.

(1) Size & Complexity: Databases can be very large and complex making manual comparison difficult, laborious, time-consuming and highly error-prone. Further, comparison may have to be repeated many times.

(2) DBMS Dependence: Both databases under comparison need to be viewed or scanned simultaneously. If this is performed using the host database management system (DBMS) or design tool, two licences for that tool are unavailable for normal use.

(3) Portability: Some common format is required in which to present the data contained in the databases for comparison, as the DBMS and comparison tool may be remote from each other or hosted on platforms of different type. Indeed, the two databases themselves may originate from tools running on different platforms. This format needs therefore to be non-proprietary and platform-independent. Ideally, the comparison tool ('comparator') also needs to be non-proprietary and platform-independent.

(4) Conformity: The data model or schema representing the databases needs to be complete, unambiguous, non-self-contradictory, and referentially integral.

(5) Knowledge of Schema: Importantly, the comparator needs information regarding the number, types and attributes of, and relationships between entities so that it can make sense of the data and decide how to handle them. In other words it needs information regarding the schema applying to the pair of databases.

(6) Data Structures: The comparator needs internal data structures not only (a) to hold the data for comparison but also (b) to hold status and control information and comparison management information and (c) to hold configuration data (values of all settable options). The design of these structures needs to be such as to facilitate comparison and reporting.

(7) Options: As databases may be very large and only some of the data may be of immediate interest to the user, it is highly desirable to offer, store and recall user-selectable options for comparison, reporting and report formatting. See 4.2.5 'The Comparison and Reporting Process' below.

(8) Discrimination: In order validly to compare a pair of entities from two databases, truly corresponding entities need to be identified. This requires some means of discrimination between entities. Therefore (a) entity types require at least one unique characteristic and (b) entities of any particular type require unique values of at least one characteristic. (a) In a fully-normalised RDB there is no duplication of entity types and the entity type (table name) provides the discriminator but for (b) there is generally no reliable discriminator upon which pairing can be based. Although all entity types have a key field (usually some form of 'id' field) containing individually unique values within that entity type, where the newer version of the database has not been created from the older version by direct modification within the RDBMS, the key field values for the two entities truly forming a pair cannot be relied upon and may well be completely different. This requires that some form of discriminator other than the key field be available for use as an alternative.

(9) Sign-off: The results of performing a comparison between two database versions are a valuable asset in areas such as configuration management and quality assurance, and are probably essential to certification. Some means of certifying and time-stamping the results along with a record of all identifying parameters, configuration and selected options is needed.

(10) Speed: The time required for the comparison of two database varies exponentially as the sizes of the databases. Essentially there are two time factors: the time taken to pair up corresponding objects and that taken to compare their characteristics. Although the second factor varies linearly with the numbers of objects, the first follows a law which is complex but is related to the product of the sizes of the two databases and hence basically a square law. In other words, potentially, it takes roughly four times as long to compare databases of twice the size. The overall law for the types of database under consideration is somewhere between $2n(n + 0.5)$ and $2n(n + 1)$ for which the time ratio approaches a square law for large databases of equal size. Some means of speeding things up is very important.

(11) Independence of the Tool from the Schema: The tool should be able to handle databases having different schemas.

4 The Invention: The Differencing Engine

This invention relates to the field of relational databases (RDBs) held in electronic form usually created and maintained with the use of an RDB management system (RDBMS). More particularly, but not exclusively, it relates to databases which are represented by fully-normalised schemas (or schemae).

The purpose of the invention is to compare two versions of a database of the above type, where both versions conform to the same schema, and determine the differences between them fully and accurately. For convenience we refer to the two database versions as the 'older' and 'newer'. This invention is referred to below as 'the tool'.

During comparison, data entities of like type in the two databases are paired by value of a particular field such as 'name' or 'id' (identity number) whose field values are expected to be individually unique within that table. Thus those entities in one database which correspond to (potentially 'modified') entities in the other, and those which exist only in either the older or newer ('deleted' and 'inserted' entities, respectively) are identified. This particular field is known as the discriminator and may or may not be a 'key' field in normal database terminology.

The user is offered a choice of discriminators. Where the discriminator is not a key field and values in the discriminator are not unique within a table (ie. for a particular entity type), objects are paired by considering their positions in the two entity hierarchies; only if they have identical 'ancestry' are they considered a pair. As it is not known in advance whether or not all values of the chosen discriminator are unique for any one entity type, all potential entity pairs are checked for identical 'ancestry' in terms of their ancestor's discriminator values.

Potentially, paired (common) entities differ or are the same, difference being determined by comparing the values of all attribute pairs and a pair of entities being classed as similar if no attribute pair has differing values. Differing and similar entities are referred to as 'modified' and 'unmodified', respectively.

Hence, during comparison all similarities and differences are thus identified.

During the pairing process, every complete and incomplete data entity pairing is represented by a special entity called a difference entity which holds process management information and, most importantly, references to the data entities.

Because the databases are constrained to conform to a schema, an object known as the schema object is made available to the tool and from this object the tool determines what entity types exist and what attributes each type has. The tool is then able to present options to the user which allow him to select which entity type or types to compare, which similarities and/or differences to report, and how to

report them. This includes optional reporting of unmodified, modified, deleted or inserted entities, and (for modified entities) their attribute value changes. Additional options, specific to each entity type, allow the user to select which attribute or attributes to compare for value changes. For each entity, its position in the heirarchy is reported.

Where entities have a characteristic, normally a description, whose values contain embedded coded data the tool allows the user selection of subsets of this data and/or its bedding via further options. A typical case might be: "Number of birds spotted this week <ringCategory = "raptor"/>". The data item 'ringCategory' in this case is coded by putting it in a recognisable form, here an XML-type tag format, and embedding it into the description text. The user can optionally retgard or ignore either or both of the text ('bedding') and coded ('embedded') parts of the description whilst comparing descriptions.

It may be that an entity in one database has a truly corresponding entity in the other but has not been paired with it due to the discriminator value having changed, resulting in it being classed as a deletion and its partner as an insertion. If the discriminator is some thing like a name and names of entities have been changed (perhaps to make them more meaningful) there may be many unrecognised pairs. The tool gives the user the option of applying various 'fuzzy' (approximate) matching methods to each pair of discriminator values. These methods are parametrised to allow control of the degree of approximation to be applied by the chosen method.

One of the two methods adopted (from several available) uses the so-called 'Levenstein edit distance' criterion ('distance'). The other uses an approximation algorithm which essentially looks for the appearance in one string of character sequences (not necessarily contiguous) contained in the other string and returns an 'approximateness' figure expressed as a percentage. *[Is 100% 'approximateness' equivalent to equality or to total dissimilarity? Now, there's a conundrum. (I would assert the former.)]*

4.1 The Solution: Addressing the Issues

The requirements implied in 'The Problem: Issues to be Addressed' above are met in the following corresponding ways.

(1) Size & Complexity: Automate the process(es) using software.

(2) DBMS Dependence: Provide a standalone tool that is able to input and hold data obtained from both databases and operate on the data independently of the originating tools.

(3) Portability: Use a textual file format that preserves data structures and relationships, such as XML (Extensible Markup Language), and choose a character encoding that is supported by the largest majority of platforms, such as ISO-8859-1 or UTF-8.

(4) Conformity: Make it a requirement that databases for comparison adhere to a standard schema and that the schema is fully-normalised. This usually requires that a thorough object-oriented analysis be carried out as part of the schema design. This constraint can be relaxed under the intersection conditions discussed in 'Schema Changes' above (2.3.2).

(5) Knowledge of Schema: Provide and use an object containing this information in a form easily accessible and meaningful to the comparator. This is referred to as the schema object.

(6) Data Structures: Use (a) two sets of object-oriented data structures to hold the data for comparison, (b) a set to hold status and control information and comparison management information, and (c) a further set to hold the configuration data (values of all settable options).

(7) Options: Provide a means of offering, storing and recalling user-selectable options for comparison, reporting and report formatting. This may take the form of a user-composable file for command-line operation and a combination of this and a graphical user interface (GUI) for interactive operation together with optional saving of changed option settings. Again, availability of the schema object facilitates provision of schema-dependent user-selectable options. See 4.2.5 'The Comparison and Reporting Process' below.

(8) Discrimination: (a) Restrict usage of the tool to RDBs and (b) adopt a discrimination strategy as follows. Offer the user a choice of discriminator including one based on key values. Where key values can be relied upon to be unique recommend use of the discriminator based on these. Where they cannot be relied upon, where the discriminator is not a key field, and where values in the discriminator are not unique within a table (ie. for a particular entity type), objects are paired by considering their positions in the two entity hierarchies; only if they have identical 'ancestry' (root and branch) are they considered a pair. As it is not known in advance whether or not all values of the chosen discriminator are unique for any one entity type, all potential entity pairs are checked for identical 'ancestry' in terms of their ancestor's discriminator values.

(9) Sign-off: Include identification, configuration, option-selection, and time-stamp information automatically with the recorded results.

(10) Speed: The objects being paired and compared actually reside in structures (in this case, trees) implied by their referential relationships. This means that the time-consuming problem of pairing can be solved by applying the 'set theory' solution to deep searching of trees. By deriving the 'position' of an object in its implied structure, ie. by tracing its referential 'ancestry' (the definition of its branch of the tree or root-to-object path), and hashing this positional information for use as a key in a hash, we can derive a table of object references against position hashes. This speeds up pairing enormously as, once we know the position of an object in one set and provided we have separate hashes for the two sets of objects, we can determine very quickly if another object exists in the corresponding position in the other set.

(11) Independence of the Tool from the Schema: This is possible within the constraints discussed in 2.3.2 'Schema Changes' above and is accomplished by generating all schema specific structures and functions from the schema object.

4.2 Implementation: Easi-Diff (The Differencing Engine)

[At the end of each paragraph or sub-paragraph below the relevant 'Solution' paragraph number in given in parenthesis.]

The tool is operable in two modes: command-line only mode and GUI (Graphical User Interface) mode. It is fully automatic in the former. In the latter it is essentially automatic and processing is only interrupted to allow the user to select options. (1) It operates completely independently of the originating DBMSs. (2)

In this particular implementation the underlying schema is proprietary and is known as the Beach Solutions Easi-Studio Schema ("the Schema"). (4) Note that extensions to this schema meet the conditions discussed in 'Schema Changes' above (2.3.2) and any degree of intersection is valid.

A common textual file format is chosen in which to represent the 'data' (data and meta-data) read from the 'older' and 'newer' databases (database versions) for comparison. Any portable (platform-independent), commonly-used and easily-parsable format which preserves all data is acceptable. Of the few available, XML is considered the ideal and is used by the tool. (3)

Further, the chosen implementation language, Perl is one of many (another being Java) that are platform independent, or at least as platform-independent as any. (3) Perl has a number of other features which stem form its origins as a textual data mainpulation language and make it emminently suited to the task. (1) The GUI aspects are coded in an extension of Perl known as Perl/Tk.

4.2.1 The Data Preparation Process

The data are read from each database into two corresponding XML files using whichever proprietary tools are available with the particular RDBMS. With legacy Beach Solutions Easi-Studio databases based on Microsoft Access, Beach Solutions Easi-Studio Script Services ("Script Services"). (3)

In future the native database format underlying all Beach Solutions' tools will be XML and no data preparation will be necessay regarding Beach Solutions databases.

4.2.2 The Configuration Process

In either mode of operation the first action taken is to parse the XML file containing the default (or most recently saved) user options. This textual file is known as the configuration file ('config file') and can be created by hand in any text editor or XML-compliant editor (such as XML-Notepad). It is viewable by any XML-compliant browser such as Internet Explorer 5.x.

Of several possible parsers the one chosen is Perl's built-in XML::Parser known as ExPat. The adoption of one of the available DOM parsers was declined due to their memory-hungry nature (although they do offer distinct advantages for parsing generic XML files, which is not a concern here).

As the config file is being parsed the parser instantiates the required configuration objects on the fly and populates them with the various option values. In the GUI mode only, the graphical option-selectors are set in accordance with the option values from the configuration objects and the GUI is then displayed.

4.2.3 The Data Input Process

Prior to parsing, the XML database files are checked for the presence of characters with an encoding incompatible with the parser, any such character being translated to an equivalent character with compatible encoding.

These files are then parsed into a form that facilitates inspection and comparison. Of the several possible forms, an object-oriented structure is used, with object classes corresponding to the database tables ('entity types') and the object class characteristics corresponding to the fields ('entity attributes'). These object classes are generated from the schema object thus making the Difference Engine core independent of the database schema (11).

During parsing of an XML file an object class of appropriate type is instantiated for each record ('entity') of its associated table, the characteristic values for that object are set to the corresponding field values for that record and a reference to each instantiated object is added to one or other of two object lists: one each for the older and newer databases respectively. At the same time, for speed and efficiency of later search, a reference to the object is also saved in a hash keyed by object identity. (6)(a)

4.2.4 The Pairing Proc ss

A set exists of difference-object classes of types corresponding to database entity types (one-to-one) and having attributes that allow reference and control data to be stored during operation of the tool. This object-class set is produced from the common database schema using the information contained in the schema object. (5),(6)(b). It is generation of this object-class set that makes the Differencing Engine core indiepender of the database schema (11).

The list of 'old' (older) data objects is scanned, a difference object of appropriate type is created for the old object (ie. its class is instantiated), a reference to the old object is stored on the associated difference object, a reference to the new difference object is placed in the list of difference objects, and the old object is marked as having been 'touched' (visited). This list consequently contains objects which potentially are unmodified, modified or deleted.

At the same time, the ancestral line ('lineage') of the old object is traced and recorded as an ordered list. A lineage is essentially a path or branch of a tree. In this particular implementation, the list is in most-ancient-first (ie. hierarchically 'top down') order and is implemented as a string of symbol-separated values. The values used are those of the chosen discriminator, for each ancestral entity type, and for lineage-matching convenience includes the type for the object itself. (8) Note that the lineage could be implemented in one of several other ways, eg. a least-ancient-first (ie. hierarchically 'bottom-up') set of values in an array.

For speed and efficiency of search for comparing lineage when later attempting to pair objects, a reference to the difference object in hand is saved in a hash keyed by lineage (in this case, the old object's lineage). This is vital to the efficiency (and hence speed) of pairing and is what makes the Differencing Engine so fast (10).

Then the list of 'new' (newer) objects is scanned. This list contains objects which potentially are unmodified, modified or inserted. For each new object, the ancestral line ('lineage') of the object is traced and recorded as an ordered list, and the object is marked as having been 'touched'. A check is made for the existence of a difference object with this lineage by looking for a defined reference to a difference object in the hash of lineages (10). If one exists, this difference object corresponds to (and has a reference to) an old object having the same lineage and a potential partner has been found. The type of the old object and the value of its discriminator are retrieved and, if they agree with those of the new object under consideration, an actual partner has been found. A reference to the new object is stored in the associated difference object.

If no actual partner has been found, a new difference object of appropriate type is created (its class is instantiated) for the new object, a reference to the new difference object is placed in the list of difference objects, and a reference to the new object is stored on the new difference object. This completes the pairing process.

4.2.5 The Comparison and Reporting Process

The comparison process itself follows. For efficiency, reporting is carried out as part of this process (ie. 'on the fly').

The list of difference objects is scanned and for each difference object its references to old and new objects are inspected for definition. If only one is defined, the object referred to has either been deleted (in the case of an old object) or inserted (in the case of a new object). Its details are retrieved and the deletion or insertion as appropriate is reported.

If both references are defined, the objects referred to have been paired but may or may not be involved in a modification. The pair of values is compared for each and every attribute on these

objects. If the values of no pair differ the object is reported as unmodified. If the values of any pair differ the object is reported as modified and all attribute value changes are reported.

If the attribute is one whose values contain embedded coded data, this is filtered in accordance with the user's option selections before reporting. (7)

Throughout execution of the comparison and reporting process the user's option selections are taken into account: these options fall into five groups: (7)

(1) Discriminant: Typically, compare by 'name' or by 'id'

(2) Difference Class: Report unmodified, modified, deleted or inserted objects, or any combination. Where modified, report attribute value changes or not

(3) Difference Type: Report to include one or any combination of the existing entity types

(4) Filtering: Report embedded coded data and/or its bedding, or neither

(5) Reporting Format: Report in one or more of the available implemented formats formats (eg. TSV (text), CSV, RTF, HTML, MIF, etc)

User option selections can be saved in XML format, optionally overwriting the default configuration file (7)

4.3 Summary of Key Features

- Compares two relational databases in XML file format
- Provides independence from the originating DBMS
- Independence of Differencing Engine core from schema
- Uses a discriminator (matching characteristic) for pairing objects for comparison
- Offers the user a choice of 'discriminator' additionally to table keys
- Handles any degree of intersection between databases for comparison
- Holds database entity data for comparison in data objects
- Uses difference objects to hold status, control and management data
- Pairs corresponding data objects by type, discriminator value, and position in database hierarchy ('ancestry') to ensure valid matching
- Operates at high speed even for large databases through employment of efficent hashing algorithms
- Operates in command-line-only or GUI mode
- Provides a variety of user-selectable options

- Allows fine selection of which elements & attributes are compared
- Allows selection of which types of change are reported (eg. deletions)
- Allows selectable filtering of description field text & custom tags
- Selectably provides various difference report file formats
- Offers colour-coded report option
- Provides fuzzy matching options

- Accepts option selections via a configuration file in XML format
- Allows saving of GUI-modified option selections in configuration file
- Holds configuration options data in object-oriented structures

5 Appendix A: Normalisation

'Normalisation' formally consists of five tests that are applied to the set of entities and their sets of attributes. These tests are called the Normal Forms (NFs).

It goes to making the 'meaning' of a database clear. This is achieved by formal analysis and a sound methodology such as entity-relationship modelling, whose objectives are:

- Removing redundancy & inconsistency = avoiding duplication of data, putting 'one fact in one place', and putting the right fact in the right place
- Ensuring the real world is correctly & adequately represented by the entity types & attributes used
- Ensuring the correct relationships exist

The database should be fully normalised and should preferably be normalised with respect to the 1st to 5th normal forms, inclusive, viz:

- 1NF: There are no repeating attributes or groups of attributes on any entity.
- 2NF: Where a primary key on an entity is a composite key (ie. consisting of a group of the attributes taken together), all non-key attributes on that entity are dependent on the whole of the primary key (on all attributes in the group).
- 3NF: There are no functional interdependencies between non-key attributes on any one entity, ie. the value of one (non-key) attribute on the entity should not depend on the value of another.
- 4NF: Where multi-value dependencies exist within a compound key they are functional dependencies. ie. attributes of more than one entity type must not appear in the same table.
- 5NF: No non-trivial non-loss decompositions exist. (The only non-loss decompositions of a table all have a candidate key of the table as their candidate key.). ie. no entity type can be split into separate entity types without the loss of data.

Referential integrity must exist., ie. where the database in held (eg. outside of the RDBMS in, say, XML format) that the relationships between entities are complete and intact whether implemented by containment or cross-reference (the structure is not fragmented by missing or broken links, and all links are valid)

DiffEngPatentInput_2nd_draft.doc